Machine Learning Final Project Responses

Enron dataset

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?

   A: **The overall goal** is to build a "person of interest" identifier based on the public Enron financial and email dataset which was made public as a result of the scandal. Machine learning can be useful for building an identifier of this type because it can identify patterns and trends in the data that might not necessarily be apparent just by observation, and use it to predict persons of interest that might be involved in fraud. Since the Enron dataset already provided training labels of POI (person of interests) vs. non-POI's to classify the data with, a supervised training methodology is what was needed.

   In regards to the data, total number of employees were 146, with 21 categories (features), and 18 people of interest (poi). There were a few **outliers** I noticed. There were two within the "employee names" - one was named "TOTAL", probably as the result of adding together the results, and another named "THE TRAVEL AGENCY IN THE PARK" which I removed. I also chose to replace NaNs and negative numbers for all the financial data that I was using in my features list with zeros. Originally, there were 51 employees with a salary of **NaN**, 64 employees with a bonus of **NaN**, and 21 **NaN** payments.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values.

   A: **The features** I chose to use were ones that I intuitively thought were of core importance to the dataset, poi, salary, bonus and total payments. The others I chose using <u>SelectKBest</u> scoring selecting the <u>10 best scores</u>, and I put this in a function called: **best_features_suggestion**. The result of this function was as follows:

   [('shared_receipt_with_poi', 8.903821557165571), ('from_poi_to_this_person', 5.446687483325353), ('loan_advances', 2.5182610445203437), ('from_this_person_to_poi', 2.470521222656084), ('to_messages', 1.7516942790340737), ('total_payments', 0.3496271530428018), ('exercised_stock_options', 0.22826733729104948), ('total_stock_value', 0.1661191232097667), ('from_messages', 0.1587702392129193), ('from_messages', 0.1587702392129193), ('bonus', 0.07794885577722988), ('restricted_stock', 0.031333216297618496), ('expenses', 0.01397841382175243), ('salary', 0.00016005424569618 4)].

   I **did not implement any scaling**, however I tested my data using the MinMaxScaler to see if it made a positive difference in the predictability of the data, but it didn't appear to make much difference at all, so I commented it out of my code.

   I then **engineered two new features** that I added to my features_list, called <u>fraction_from_poi</u>, <u>fraction_to_poi</u> that is a proportion created by dividing from messages and to messages by the total of all messages. Once adding them to my features_list I ran my **best_features_suggestion** function again so I could further refine my list. Upon reviewing again I decided to remove from_poi_to_this_person, from_this_person_to_poi as they seem extraneous with my new features and rated fairly low when I ran the **best_features_suggestion** tool again.

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

   A: The algorithm I chose by iterating through a list of algorithms printing out the time to learn and predict, the accuracy and the f1 score of each algorithm, as well as the settings for each algorithm so that I could choose to change or add parameters to fine tune to give the best accuracy scores.

   My <u>algorithm list</u> I iterated through were **Naive Bayes, Support Vector Machine, Decision Tree and AdaBoost**.

```
classifiers = [GaussianNB(),
               SVC(kernel="rbf", C=10000.0),
               DecisionTreeClassifier(),
               DecisionTreeClassifier(min_samples_split = 40),
               DecisionTreeClassifier(min_samples_split=40, max_depth=4),
               AdaBoostClassifier(),
               AdaBoostClassifier(
                   DecisionTreeClassifier(min_samples_split=20, max_depth=4),
                   algorithm="SAMME.R",
                   n_estimators=60,
                   random_state=0)]
```

   Ultimately, the last AdaBoostClassifier in this list is what I choose because it had the best accuracy and f1 score. I found it interesting how some of my classifiers had a high accuracy, however the f1 score was low - or vice versa. This means that in some cases the algorithms ranked the data okay or not okay, but then the f1 score says that it didn't do very well on the rest of the test set creating false positives.

4. What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier).

   A: When you tune the parameters of an algorithm, you are tuning the settings of the model to discover the most likely predictions. If you don't do this well, the result could be that future data inputs for a machine to learn from won't be properly placed for prediction. This could result in bad decision making.

   AdaBoost had the best performance, but unfortunately it does not support estimators like GridsearchCV to help automate picking the best parameters. Instead I did some research of the classifier itself and tried **different combinations to tune my classifier**. I chose to add Decision Tree parameters to AdaBoost as well so I had some control over the decision stumps. A weak classifier (decision stump) is prepared on the training data and through several iterations it finds the best classifier through weighted scores of predictability.

Here is **what I chose to use**:

AdaBoostClassifier(DecisionTreeClassifier(min_samples_split=20, max_depth=4), algorithm="SAMME.R", n_estimators=60, random_state=0)

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

   A: Validation is the process where a trained model is evaluated with a testing data set - creating a separate training vs. testing set. But unfortunately there is a trade-off:
   If you increase your test set, you get better validation but less learning result, and vice versa.

   In order to validate my analysis, I had to experiment with changing the size of the testing set. When I changed the testing set from .3 to .4 proportionally, I noticed that I did get better accuracy scores. However the preciseness of predicting the test data went down.

6. Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance.

   A: The 2 evaluation metrics I used were recall and precision. **Recall** basically says, out of all the items that are truly positive, how many were actually correctly classified as positive? Or simply put, it "recalls" how an individual concept is doing. **Precision** says, out of all the items labeled positive, how many truly ARE positive and belong to the positive class? In other words, how "precise" was the prediction?

   For my test, set, here were the evaluation metrics:
   POI's in the test set: 3
   Number of true positives: 2
   Recall: 0.667
   Precision: 1.000
   f1 score: 0.800

   So in this case 67% of my test set were predicted correctly as a POI (recall) actually was a POI, there were 2 true positives. In regards to precision, 100% of my predicted test set were classified correctly. f1 score is the average of recall and precision. f1 score was a useful tool in finding a good classifier.

   Using **tester.py**, the results was a bit different due to the different size of the testing/training.
   Accuracy: 0.86300
   Precision: 0.47899
   Recall: 0.31350
   F1: 0.37897
   F2: 0.33677
   Total predictions: 15000